



ELSEVIER

Computational Geometry 15 (2000) 69–89

Computational
Geometry

Theory and Applications

www.elsevier.nl/locate/comgeo

Fast and accurate collision detection for haptic interaction using a three degree-of-freedom force-feedback device

A. Gregory, M.C. Lin*, S. Gottschalk, R. Taylor

Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC 27599-3175, USA

Abstract

We present a fast and accurate collision detection algorithm for haptic interaction with polygonal models. Given a model, we precompute a hybrid hierarchical representation, consisting of uniform grids (represented using a hash table) and trees of tight-fitting oriented bounding box trees (OBBTrees). At run time, we use hybrid hierarchical representations and exploit frame-to-frame coherence for fast proximity queries. We describe a new overlap test, which is specialized for intersection of a line segment with an oriented bounding box for haptic simulation and takes 42–72 operations including transformation costs. The algorithms have been implemented as part of H-COLLIDE and interfaced with a PHANTOM arm and its haptic toolkit, *GHOST*, and applied to a number of models. As compared to the commercial implementation, we are able to achieve up to 20 times speedup in our experiments and sustain update rates over 1000 Hz on a 400 MHz Pentium II. In practice, our prototype implementation can accurately and efficiently detect contacts between a virtual probe guided by a force-feedback arm and large complex geometries composed of tens of thousands of polygons, with sustained KHz rates. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Collision detection; Haptic rendering; Hybrid representation; Virtual contact determination

1. Introduction

Research in computational geometry provides an algorithmic foundation and analytic tools for geometric problems encountered in many fields of the arts, sciences and engineering. In this paper, we describe applications of geometric concepts and data structures, namely spatial partitioning and hierarchical representations, to the problem of collision detection (another well-studied topic in computational geometry) for haptic interaction with virtual environments.

1.1. Motivation

Virtual environments require natural and real-time interaction between computer systems and users. Compared to the presentation of visual and auditory information, methods for haptic display are not

* Corresponding author. [Http://www.cs.unc.edu/~geom/HCollide/](http://www.cs.unc.edu/~geom/HCollide/)

E-mail address: lin@cs.unc.edu (M.C. Lin).

as well developed. Haptic rendering as an augmentation to visual display can improve perception and understanding both of force fields and of world models populated in synthetic environments [6].

Algorithmic techniques and geometric engineering are required to integrate force feedback devices with 3D graphical display systems to offer new avenues toward promising advancement in haptic interaction with the virtual environments. Possible applications include rapid prototyping, interactive visualization and manipulation of molecular structures, tolerance verification for virtual mock-up, surgical simulation, teleoperation for manufacturing and medical diagnosis, scientific exploration, personnel training and education.

Haptic display is often rendered through what is essentially a small robot arm, used in reverse. Such devices are now commercially available for a variety of configurations (2D, 3D, 6D, specialized for laparoscopy or general-purpose). The system used in this work was a 6-Degree-of-Freedom(DoF)-in/3-DoF-out SensAble Technologies PHANToM arm.

“Real-time” graphics applications have display update requirements somewhere between 20 and 30 frames/second. In contrast, the update rate of haptic simulations must be as high as 1000 updates per second in order to maintain a stable system. This rate varies with the spatial frequency and stiffness of displayed forces, and with the speed of motion of the user. Also, the skin is sensitive to vibrations of greater than 500 Hz, so changes in force at even relatively high frequencies are detectable [10].

In order to create a sense of touch between the user’s hand and a virtual object, contact or restoring forces are generated to prevent penetration into this virtual object. These forces are computed by first detecting if a collision or penetration has occurred, then determining the (projected) contact point on the object surface. Most existing algorithms address the collision detection and contact determination problems for relatively small models consisting of only a few thousand polygons or a few surfaces. Our ultimate goal is to be able to achieve smooth, realistic haptic interaction with CAD models of high complexity (consisting of tens of thousands of primitives) for virtual prototyping applications. In addition, we are aiming at designing algorithms that are easily extensible to support a wide range of force-feedback devices (including 6-DoF arms) and deformable surfaces.

There is considerable literature in computational geometry, computer graphics, robotics and virtual environments on fast collision detection between geometric models. A number of algorithms based on bounding volume hierarchies, spatial decomposition or special properties of geometric models (e.g., convex polytopes, implicitly defined surfaces) have been proposed. However, none of the existing methods are able to detect all the contacts *in less than a millisecond* on complex geometric models composed of tens of thousands of polygons *accurately* without gross approximation. Computing all contact points accurately to support real-time haptic interaction with complex virtual environments at KHz rates is one of the major challenges for geometric engineering and algorithm design.

1.2. Main contribution

In this paper we present an algorithmic framework for fast and accurate collision detection for haptic interaction. It consists of a number of geometric algorithms and a prototype system specialized for computing contact(s) between the probe of the force-feedback device and objects in the virtual environment. To meet the stringent performance requirements for haptic interaction, we use a *hybrid* approach and specialize many earlier geometric algorithms for this application. Our framework utilizes many geometric concepts and data structures from computational geometry. These include:

- *Spatial partitioning.* It decomposes the workspace into uniform grids or cells, implemented as a hash table to efficiently deal with large storage requirements. At runtime, the algorithm can quickly find the cells containing the path swept out by the hand-held probe.
- *Bounding volume hierarchy.* Bounding volume hierarchies have been well studied in computational geometry for various geometric operations. Any bounding volume hierarchy can be used with this framework. Some of the well-known examples include trees of sphere [20,35], OBBTrees [4,16], K-DOPs [22,23], SSVs [25], etc. In our prototype implementation, we used OBBTrees. An OBBTree is a bounding volume hierarchy [16] of tight-fitting oriented bounding boxes (OBBs). For each cell consisting of a subset of polygons of the virtual model, we pre-compute an OBBTree. At run-time, most of the computation time is spent in finding collisions between an OBBTree and the path swept out by the tip of the probe between two successive time steps. To optimize this query, we have developed a very fast specialized overlap test between a line segment and an OBB, that takes as few as 42 operations and only 72 arithmetic operations in the worst case, including the cost of transformation.
- *Frame-to-frame coherence.* Typically, there is little movement in the probe position between successive steps. The algorithm utilizes spatial coherence by caching the contact information from the previous step to perform incremental computations.

The algorithm pre-computes a hybrid hierarchy. Our algorithmic framework also allows the application program to select only a subset of the approaches listed above.

We have successfully implemented all the geometric algorithms described above, interfaced them with *GHOST* (a commercial haptic library) [39] and used them to find surface contact points between the probe of a PHANTOM arm and large geometric models (composed of tens of thousands of polygons). Their performance varies based on the geometric model, the configuration of the probe relative to the model, machine configuration (e.g., cache and memory size) and the combination of techniques used by our system. The overall approach results in a factor of 2–20 speed improvement as compared to earlier algorithms and commercial implementations. For a number of models composed of 5,000–80,000 polygons, our prototype implementation is able to sustain a KHz update rate on a 400 MHz Pentium II.

The results presented in this paper are specialized for a point probe against 3D object collision detection. We conjecture that it can be extended to compute object-object intersection for a six-degree-of-freedom haptic device as well.

1.3. Organization

The rest of the paper is organized in the following manner. Section 2 provides a brief survey of related research. Section 3 describes the system architecture and algorithms used in the design of our system. We discuss the implementation issues in Section 4. We present analysis of our algorithm in Section 5, discuss our experimental results and compare their performance with a commercial implementation in Section 6. To conclude, we discuss possible future directions of this research.

2. Related work

Collision detection and contact determination are well-studied problems in computational geometry, computer graphics, robotics and virtual environments. Due to limited space, we refer the readers to [19, 26] for recent surveys. In the ray-tracing literature, the problem of computing fast intersections between

a ray and a three-dimensional geometric model has also been extensively studied [1]. While a number of algorithms have been proposed that make use of bounding volume hierarchies, spatial partitioning or frame-to-frame coherence, there is relatively little available on hybrid approaches combining two or more such techniques.

Bounding volume hierarchies. A number of algorithms based on hierarchical representations have been proposed. The set of bounding volumes includes spheres [20,35], axis-aligned bounding boxes [5,19], oriented bounding boxes [4,16], approximation hierarchies based on S-bounds [7], spherical shells [24] and k-dop's [22,23]. In the close proximity scenarios, hierarchies of oriented bounding boxes (OBBTrees) and higher-order bounding volumes appear superior to many other simple bounding volumes [16].

Spatial partitioning approaches. Some of the simplest algorithms for collision detection are based on spatial decomposition techniques. These algorithms partition the space into uniform or adaptive grids (i.e., volumetric approaches), octrees [38], k-D trees or BSP's [32]. To overcome the problem of large memory requirements for volumetric approaches, some authors [34] have proposed the use of hash tables.

Utilizing frame-to-frame coherence. In many simulations, objects move only a little between successive frames. Many efficient algorithms that utilize frame-to-frame coherence have been proposed for convex polytopes [3,8,27]. Cohen et al. [9] have used coherence-based incremental sorting to detect possible pairs of overlapping objects in large environments.

Research in haptic rendering. Several techniques have been proposed for integrating force feedback with a complete real-time virtual environment to enhance the user's ability to perform interaction tasks [10, 11,13,28,29,33,40]. The commercial haptic toolkit developed by SensAble Technologies, Inc. also has a collision detection library [37,39]. Ruspini et al. [36] presented a haptic interface library "HL" that uses a multi-level control system to effectively simulate contacts with virtual environments. It uses a bounding volume hierarchy based on sphere-trees [35].

Nahvi et al. [31] designed a haptic display system for manipulating virtual mechanisms derived from a mechanical CAD design. It uses the Sarcos Dexterous Arm Master and Utah's Alpha_1 CAD system, with algorithmic support from a tracing algorithm and a minimum distance framework developed by Johnson, Cohen et al. [21,41]. They utilized a variety of algorithmic toolkits from RAPID [16] to build an OBBTree for each object, Gilbert's algorithm [15] to find distances between two OBB's and a tracing algorithm for parametric surfaces. Their system takes about 20–150 milliseconds for models composed of 500–23,000 triangles on an SGI Indigo2 and 4 milliseconds for models 5 composed of 3 parametric surfaces on a Motorola 68040 microprocessor.

Gibson [14] and Sobierajski [2] have proposed algorithms for object manipulation including haptic interaction with volumetric objects and physically-realistic modeling of object interactions. Researchers at Boeing proposed a voxmap-sampling technique [30] for 6-dof haptic rendering, where point samples from one surface are tested against the voxel representations of the static environment. This approximation approach achieves constant query time in exchange for accuracy and correctness of haptic force display. False collisions are often reported and actual collisions are occasionally missed, due to discretization of the workspace and sampling of the moving object.

3. Fast proximity queries for haptic interaction

In this section, we describe the haptic system setup and algorithmic techniques that are an integral part of the collision detection algorithmic framework for haptic interaction, H-COLLIDE.

3.1. Haptic system architecture

Due to the stringent update requirements for real-time haptic display, we run a special stand-alone haptic server written with the VRPN library available at

<http://www.cs.unc.edu/Research/nano/manual/vrpn>

on a PC connected to the PHANTOM. The client application runs on another machine, which is typically the host for graphical display. Through VRPN, the client application sends the server the description of the scene to be haptically displayed, and the server sends back information such as the position and orientation of the PHANTOM probe. The client application can also modify and transform the scene being displayed by the haptic server.

3.2. Algorithm overview

Given the last and current positions of the PHANTOM probe, we need to determine if it has passed through an object's surface, in order to display the appropriate force. The probe movement is usually small due to the high haptic update rates. This implies that we only need to check a relatively small volume of the workspace for geometric contacts.

Approaches using spatial partitioning seem to be natural candidates for such situations. For large and complex models, techniques based on uniform or adaptive grids can be implemented more efficiently using hash tables. However, to achieve the desired speed, these approaches still have extremely high storage requirements even when implemented using a hashing scheme.

Despite its better fit to the underlying geometry, any hierarchical bounding volume method may end up traversing trees to great depths to locate exact contact points for large, complex models. To take advantage of each approach and to avoid some of the deficiencies of each, we propose a hybrid algorithm.

Hybrid hierarchical representation. Given a virtual environment containing several objects, each composed of tens of thousands of polygons, the algorithm computes a *hybrid hierarchical representation* of the objects as part of the off-line pre-computation. The entire virtual workspace is first partitioned into coarse-grain uniform grid cells. Then, for each grid cell containing some primitives of the objects in the virtual world, an OBBTree for that grid cell is computed and a pointer to the associated OBBTree is stored using a hash table for expected constant-time proximity queries.

Specialized intersection tests. The on-line computation of our collision detection framework consists of three phases. In the first phase, “the region of potential contacts” is identified by determining which cells were touched by the probe path, using the precomputed look-up table. In the second phase, the OBB-Tree(s) in that cell are traversed to determine if collisions have occurred, using the specialized fast overlap test described later. If the line segment intersects with an OBB in the leaf node, then the third phase computes the (projected) surface contact point(s) (SCP) using techniques similar to those in [39,41].

Frame-to-frame coherence. If in the previous frame the probe of the feedback device was in contact with the surface of the model, we exploit *frame-to-frame coherence* by traversing the surface of the mesh starting with the contact witness from the previous frame. If this traversal does not yield a contact, then we proceed using the hybrid hierarchical representation of the objects.

3.3. Overlap test based on a line segment against an OBBTree

At first glance, it is tempting to use sophisticated and optimized line clipping algorithms. However, the line-OBB intersection problem for haptic interaction is a simpler one than line clipping and the environment is dynamic and consisting of many OBBs.

For haptic display using a point probe, we can specialize the algorithm based on OBBTrees by only testing a line segment (representing the path swept out by the probe between two successive steps) and an OBBTree. (The original algorithm [16] uses an overlap test between a pair of OBBs and can take more than 200 operations per test.) At run time, most of the computation is spent in finding collisions between a line segment and an OBB. To optimize this query, we have designed a very fast overlap test between a line segment and an OBB, that takes as few as 6 operations and only 36 arithmetic operations in the worst case, not including the cost of transformation.

Here we describe this specialized overlap test between a line segment and an oriented bounding box for haptic rendering. Without loss of generality, we choose the coordinate system centered on and aligned with the box – so the problem is transformed to an overlap test between a segment and a centered axis-aligned bounding box. Our overlap test uses the Separating-Axis Theorem described in [16], specialized for a line segment against an OBB.

Specifically, the candidate axes are the three box face normals (which are aligned with the coordinate axes) and their cross-products with the line segment's direction vector. We project both the box and the segment onto each of these six candidate axes, and test whether the projection intervals overlap. If the projections are disjoint for any of the six candidate axes, then the segment and the box are disjoint. Otherwise, the segment and the box overlap.

How are the projection intervals computed? First note that the center points of the segment and of the box project to the midpoints of their respective intervals as shown in Fig. 1. We can also directly compute the lengths (actually the half-lengths) of the intervals. The intervals overlap if and only if the separation between their midpoints is less than the sum of their half-lengths.

Given a direction vector v of a line through the origin, and a point p , let the point p' be the axial projection of p onto the line. The value $d_p = v \cdot p / |v|$ is the signed distance of p' from the origin along the line. Now consider the line segment with midpoint m and endpoints $m + w$ and $m - w$. The half-length of the line segment is $|w|$. The image of the segment under axial projection is the interval centered at

$$d_s = \frac{v \cdot m}{|v|}$$

and with half-length

$$L_s = \frac{|w \cdot v|}{|v|}.$$

Given a box centered at the origin, the image of the box under axial projection is an interval with midpoint at the origin.

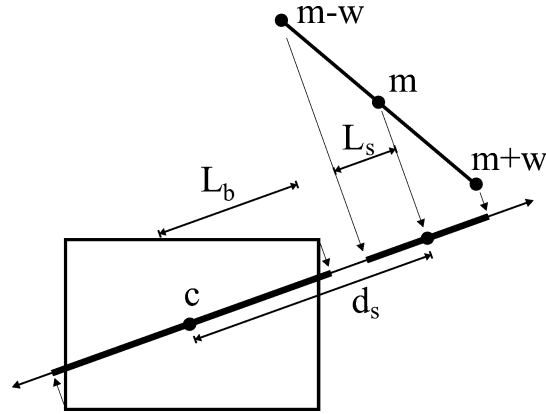


Fig. 1. Overlap test between a line segment and an OBB.

Furthermore, if the box has thicknesses $2t^x$, $2t^y$ and $2t^z$ along the orthogonal unit directions u^x , u^y and u^z , the half-length of the interval is given by

$$L_b = \left| \frac{t^x v \cdot u^x}{|v|} \right| + \left| \frac{t^y v \cdot u^y}{|v|} \right| + \left| \frac{t^z v \cdot u^z}{|v|} \right|.$$

With the intervals so expressed, the axis v is a separating axis if and only if (see Fig. 1)

$$|d_s| > L_b + L_s.$$

If we assume that the box is axis-aligned, then $u^x = [1, 0, 0]^T$, $u^y = [0, 1, 0]^T$ and $u^z = [0, 0, 1]^T$, and the dot products with these vectors become simple component selections. This simplifies the box interval length computation to

$$L_b = |t^x v_x| + |t^y v_y| + |t^z v_z|.$$

Now, recall that the candidate axis v is either a box face normal, or a cross product of a face normal with the line segment direction vector. Consider the former case, when v is a box face normal, for example $[1, 0, 0]^T$. In this case, the components v_y and v_z are zero, and the component v_x is one, and we are left with

$$L_b = t^x.$$

The projection of the line segment onto the x -axis is also simple:

$$L_s = |w_x|.$$

So, the test for the $v = [1, 0, 0]^T$ axis is

$$|m_x| > t^x + |w_x|.$$

The tests for the candidate axes $v = [0, 1, 0]^T$ and $v = [0, 0, 1]^T$ have similar structure.

The three cases where v is a cross product of w with one of the box faces are a little more complex. Recall that, in general,

$$L_b = |t^x v \cdot u^x| + |t^y v \cdot u^y| + |t^z v \cdot u^z|.$$

For the sake of concreteness, we will choose $v = w \times u_y$. Then this expression becomes

$$L_b = |t^x(w \times u^y) \cdot u^x| + |t^y(w \times u^y) \cdot u^y| + |t^z(w \times u^y) \cdot u^z|.$$

Application of the triple product identity

$$(a \times b) \cdot c = (c \times a) \cdot b$$

yields

$$L_b = |t^x(u^y \times u^x) \cdot w| + |t^y(u^y \times u^y) \cdot w| + |t^z(u^y \times u^z) \cdot w|.$$

All of these cross products simplify, because the u vectors are mutually orthogonal, $u^x \times u^y = u^z$, $u^y \times u^z = u^x$ and $u^z \times u^x = u^y$, so

$$L_b = |t^x(-u^z) \cdot w| + |t^y(0) \cdot w| + |t^z(u^x) \cdot w|.$$

And again, using the fact that $u^x = [1, 0, 0]^T$, and so forth,

$$L_b = t^x|w_z| + t^z|w_x|.$$

The half-length of the segment interval is

$$L_s = |w \cdot (w \times u^y)| = |u^y \cdot (w \times w)| = |u^y \cdot 0| = 0,$$

which is what we would expect, since we are projecting the segment onto a line orthogonal to it.

Finally, the projection of the segments midpoint falls at

$$d_s = (w \times u^y) \cdot m = (m \times w) \cdot u^y = m_z w_x - m_x w_z,$$

which is just the y -component of $m \times w$. The final test is

$$|m_z w_x - m_x w_z| > t^x|w_z| + t^z|w_x|.$$

Similar derivations are possible for the cases $v = w \times u^x$ and $v = w \times u^z$.

Writing out the entire procedure, and precomputing a few common subexpressions, we have the following pseudo code:

```

let  $X = |w_x|$ 
let  $Y = |w_y|$ 
let  $Z = |w_z|$ 
if  $|m_x| > X + t_x$  return disjoint
if  $|m_y| > Y + t_y$  return disjoint
if  $|m_z| > Z + t_z$  return disjoint
if  $|m_y w_z - m_z w_y| > t_y Z + t_z Y$  return disjoint
if  $|m_x w_z - m_z w_x| > t_x Z + t_z X$  return disjoint
if  $|m_x w_y - m_y w_x| > t_x Y + t_y X$  return disjoint
otherwise return overlap

```

When a segment and an OBB are disjoint, the routine often encounters an early exit and only one (or two) out of the six expressions is executed. Total operation count for the worst case is: 9 absolute values, 6 comparisons, 9 additions and subtractions, 12 multiplications. This does not include the cost of transforming the segment (i.e., 36 operations) into a coordinate system centered and aligned with the box.

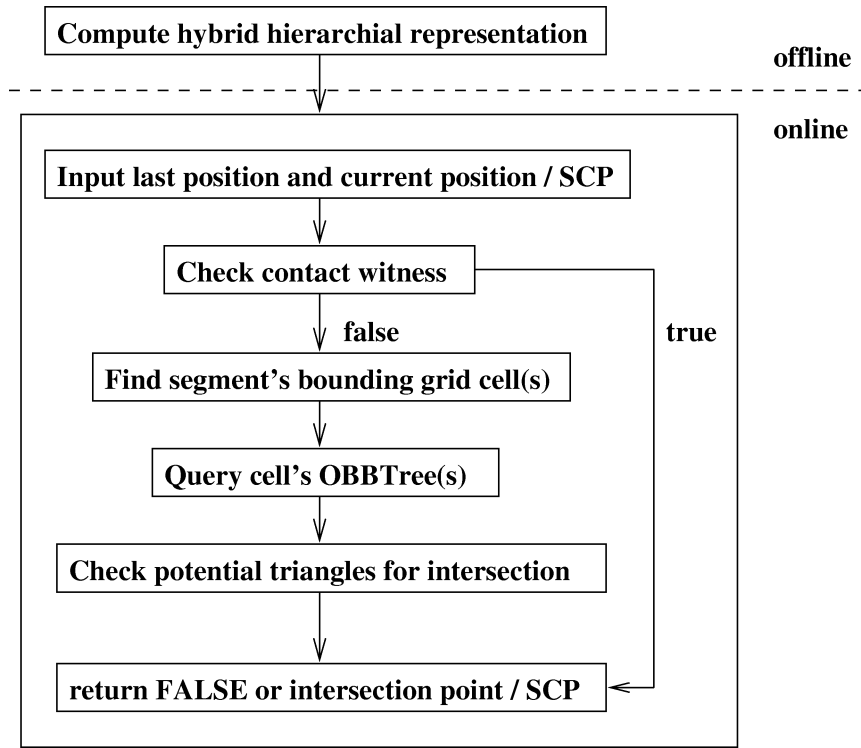


Fig. 2. The system architecture of H-COLLIDE.

4. Implementation issues

H-COLLIDE, a framework for fast and accurate collision detection for haptic interaction, is designed based on the hybrid hierarchical representation and geometric algorithmic techniques described above. Fig. 2 shows the system architecture of H-COLLIDE.

H-COLLIDE has been successfully implemented in C++. We have interfaced H-COLLIDE with *GHOST*, a commercial software developer's toolkit for haptic rendering, and used it to find surface contact points between the probe of a PHANToM arm and large geometric models (composed of tens of thousands of polygons). Here we describe some of the implementation issues.

4.1. Hashing scheme

Clearly it is extremely inefficient to allocate storage for all the cells, since a polygonal surface is most likely to occupy a very small fraction of them. We use a hash table to alleviate the storage problem. From each cell location at (x, y, z) and a grid that has len cells in each dimension, we can compute a unique key using

$$key = x + y \cdot len + z \cdot len^2.$$

In order to avoid hashing too many cells with the same pattern into the same table location we compute the actual location for a grid cell in the hash table with

$$\text{TableLoc} = \text{random}(\text{key}) \% \text{TableLength}.$$

Should the table have too many cells in one table location, we can simply grow the table. Hence, it is possible to determine which triangles we need to check in constant time and the amount of storage required is a constant factor (based on the grid grain) of the surface area of the object we want to “feel”.

Determining the optimal grid grain is a non-trivial problem in practice. We present a possible analytical solution to this issue and an accompanying analysis in the next section. Due to the practical difficulty of determining the optimal grid size, we simply set the grain of the grids to be the average length of all edges multiplied by a constant, which is chosen based on experimentation. If the model has a very irregular triangulation it is possible that there could be a large number of small triangles in a single grid cell. In this case, the triangles of each grid cell are stored in an OBBTree and checked for collision with the probe using the specialized overlap test described in Section 3.3.

4.2. User options

Since the hybrid approach used in H-COLLIDE has a higher storage requirement than either individual technique alone, the system also allows the user to select a subset of the techniques, such as the algorithm purely based on OBBTrees, to allow for better performance on a machine with less memory.

5. Algorithm analysis

In this section, we examine the runtime performance of our hybrid collision detection algorithm for haptic interaction and analyze the issues involved in determining an optimal grid size.

5.1. Runtime performance

Our algorithmic framework should be able to ensure that the running time of our approach is empirically constant, independent of the complexity of the model. Here we analyze the runtime performance of our hybrid approach, given a set of assumptions. Later in Section 6, we will see the results of relaxing some of our assumptions.

Let P be a general polygonal model representing the environment in \mathbb{R}^3 . P may not have topological connectivity. Suppose that P is given as a collection of N triangles.

Assumption 1. The last probe location is close to the current probe location such that the path swept out by the probe goes through at most S grid cells, where $S \approx 1$.

Assumption 2. P is uniformly tessellated, e.g., the number of triangles per unit surface area is constant across the surface of P .

Assumption 3. There is only one contact point with the surface of the object and one triangle in contact with the probe, since it is only the tip of the probe that is touching the surface of the virtual object.

Assumption 4. The environment is stationary and rigid.

Assumptions 1, 3 and 4 are made based on conditions often encountered in haptic rendering. Assumption 1 is normally true due to the fast update rate required for haptic display. However, Assumption 2 is not always true, especially when P is a CAD model imported into a virtual environment. However, many scanned models have uniform triangle distribution over the surface of the model. In these cases, Assumption 2 is reasonable.

Lemma 5. *Querying a preconstructed OBBTree of the model P for possible intersections with a line segment L takes $O(KH)$ time, where H is the height of a given precomputed OBBTree of P and K is the number of intersections between L and P .*

An OBBTree is a binary search tree with an OBB stored at each node of the tree. If the height of the OBBTree for a model P of N triangles is H and each OBB-line overlap test described in Section 3.3 takes $O(1)$. Therefore, querying an OBBTree takes $O(H)$ time for each intersection. If there are K intersection, then it takes $O(KH)$ time. Normally $K = 1$ given Assumption 3 in the case of haptic interaction using a three-degree-of-freedom force feedback device. And, the height of an OBBTree is $O(\log N)$ in general, but it can be $O(N)$ in the worst case.

Theorem 6. *Let N be the number of triangles in a given model P . With Assumptions 1 and 2, the expected collision query time using the hybrid data structures for possible intersections between P and a line segment L is $O(1)$.*

The basic dictionary operations using hash tables require only $O(1)$ time on average [12] and this is extended for the point location problem [34]. There are typically less than two such queries, since $S \approx 1$ given Assumptions 1 and 4. So, we can query the hash table as to which OBBTrees need to be checked in expected $O(1)$ time. According to Lemma 5, querying an OBBTree for intersection with L takes $O(KH)$ time, where H is the maximum height of the OBBTree per grid cell and K is the number of the intersections. H is $O(\log M)$ for most cases and $O(M)$ in the worst case. During the off-line computation, if the number of grid cells is G , we can bound the maximum number of triangles M per grid cell to be a small constant compared to the total number of triangles N in the model P , given Assumption 2. M can be computed by finding the maximum number of triangles per grid cell. We can also increase the number of grid cells G for larger models to obtain a constant bound on M . And, $K \ll M$. (With Assumption 3, $K = 1$.) Therefore, the overall expected running time of our hybrid approach is $O(1)$.

5.2. Determining the optimal grid size

Even with the hashing scheme, a very small grid size can consume a huge amount of memory. Furthermore, if the granularity of the grid is extremely small compared with the size of a triangle, then it can take a long time to scan that triangle into all of the grid cells it covers. (However, this is a one-time precomputation when we load the model.) Similarly, a grid size that is too large can result in too many triangles to check for intersection in one force update iteration. Many models we encounter do not have polygons of uniform sizes. Many have long and skinny triangles.

To analytically determine the optimal grid size is a rather difficult problem, since there are numerous parameters that may affect the performance of any real-time haptic systems in addition to algorithmic

issues. In order to simplify the model of computation to obtain a better intuition of the problem, we make Assumptions 1–4 in our analysis.

With this set of assumptions, let N be the total number of triangles for the object to be “felt” by the probe, M be the average number of triangles per optimal grid cell, $K = 1$ and $S = 1$. We will analyze the bounds on the number of triangles that should be contained per optimal grid cell, and determine the optimal grid size based on these numbers.

Let the cost of performing collision detection using only OBBtrees and specialized OBB-line overlap test be C_r , the cost of performing collision detection using hashing scheme alone be C_g , and the cost of using the hybrid algorithm be C_h . Let C_{obb} be the number of arithmetic operations required to perform an OBB-line overlap test, C_{tri} be the number of operations required to perform a line and a triangle intersection test, and C_1 be the number of operations required to perform the hash function evaluation for table lookup. Since the total number of nodes in an OBBTree of height h is $2h + 1$, we have

$$C_r = (2 \log N + 1) \cdot C_{obb} + C_{tri}, \quad (1)$$

$$C_g = M \cdot C_{tri} + C_1, \quad (2)$$

$$C_h = (2 \log M + 1) \cdot C_{obb} + C_{tri} + C_1. \quad (3)$$

For the hybrid method to outperform both the specialized algorithm with only OBBTrees or the uniform spatial partitioning using hash table look-up, (1) $C_h < C_r$ and (2) $C_h < C_g$ must be satisfied. Together (1) and (2) imply the following:

$$M < \frac{N}{(\sqrt{2})^{C_1/C_{obb}}}, \quad (4)$$

$$M > 2 \cdot \log M \cdot \frac{C_{obb}}{C_{tri}} + \left(1 + \frac{C_{obb}}{C_{tri}}\right). \quad (5)$$

M is bounded above and below by these two inequalities. Inequality (4) is almost always true, when we assume that M is much smaller than N . Inequality (5) states that as M increases over some minimum threshold value the inequality is better satisfied. This implies that the hybrid algorithm will only be more efficient than the specialized algorithm using the OBBTrees alone, if the average number of triangles per grid is greater than some constant.

In our implementation, C_{tri} varies from 18 to 55 operations and C_{obb} varies from 42 to 72 (including the cost of transformation), since both tests have a sequence of conditional statements that may not be executed all the time depending the contact configuration. C_1 depends on the number of operations taken by the random number generator. For the purpose of this analysis, we will assume C_1 to be in the range of 10–20 arithmetic operations. So, the ratios C_1/C_{obb} and C_{obb}/C_{tri} range from 0.9–5.5 and 0.764–4.0, respectively. This makes it rather difficult to determine one unique value for the optimal grid cell for all scenarios – even under the given set of assumptions, though in theory we can compute a range of near-optimal grid sizes based on these assumptions.

Let us now examine the impact of our assumptions on the computation of grid size. If Assumptions 1 and 3 are not entirely true, e.g., the line segment swept out by the probe occupies multiple grid cells or there are multiple triangles intersecting the line segment, then small constants will be introduced to various terms in Eqs. (1)–(3). If parts of the objects or the objects themselves start to move around, then we will also need to consider the cost of updating the data structures, which may vary depending on which part of models need to be checked for collision detection.

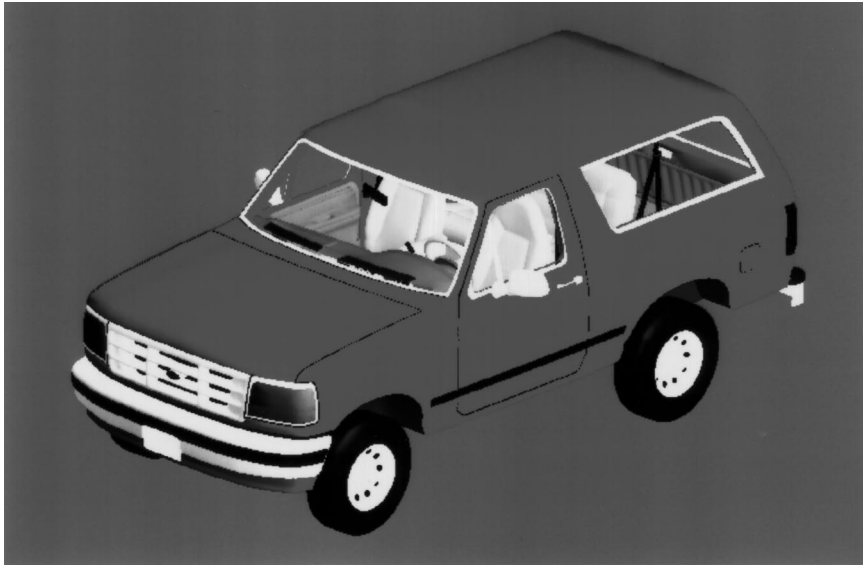


Fig. 3. Ford bronco model, 18,439 triangles.



Fig. 4. Man symbol, 5,276 triangles.

Assumption 2 is not always true in many of the CAD/CAM models. Several real-world models we have observed and used in our benchmark (see Figs. 3–7) have uneven surface tessellations. In fact, their triangle shapes and sizes vary. Many long and skinny triangles each cover several grid cells; while grid cells of a different part of the same model may have many small triangles. It is rather challenging to analytically compute *one* near-optimal grid size that actually works well in practice for *all* input models.

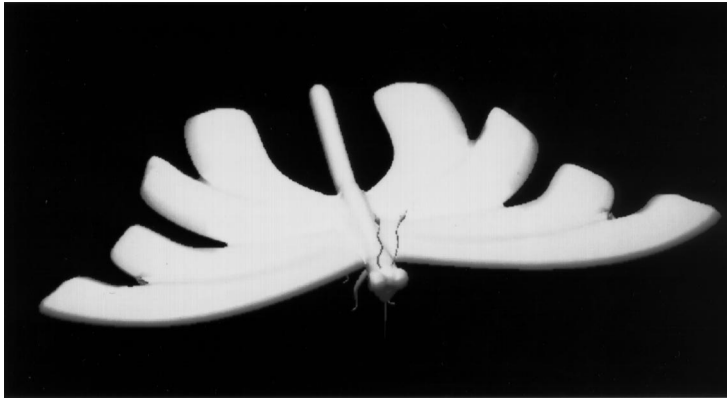


Fig. 5. Butterfly, 78,846 triangles.



Fig. 6. Man with hat, 7,152 triangles.

6. Performance and comparison

For comparison, we have implemented adaptive grids, our hybrid approach and an algorithm using only OBBTrees and the specialized overlap test described in Section 3.3. We have applied them to several models of varying complexity (see color plates at <http://www.cs.unc.edu/~geom/HCollide/model.pdf> or Figs. 3–7). Their performance varies based on the models, the configuration of the probe relative to the model, machine configuration (e.g., cache and memory size), and the combination of techniques used by our system. Our hybrid approach results in a factor of 2–20 speed improvement as compared to a native *GHOST* method for the models we have tried. For a number of models composed of 5,000–80,000



Fig. 7. Scott Paulson uses the nanoWorkbench to examine a cleaved and bent particle of tobacco mosaic virus on a representative surface. The nano surface we tested has 12,482 triangles.

polygons, our system is able to compute all the contacts and response at rates higher than 1000 Hz on a 400 MHz Pentium II.

6.1. Obtaining test data

We first obtained the test data set by deriving a class from the triangle mesh primitive which comes with SensAble Technologies' *GHOST* library, version 2.0 beta. This records the start and the endpoint of each segment used for collision detection during a real force-feedback session with a 3-DOF PHANTOM arm. We then implemented the three techniques mentioned above to interface with *GHOST* for comparison with a native *GHOST* method, and timed the collision detection routines for the different libraries using the data from the test set. The test set for each of these models contains 30,000 readings. In order to gain a better comparison between the different techniques, the frame-to-frame coherence method described in Section 3.2 was not used for these timings.

The distinction between a collision and an intersection shown in the tables is particular to *GHOST*'s haptic rendering. Each haptic update cycle contains a "collision" test to see if the line segment from the last position of the PHANTOM probe to its current position has intersected any of the geometry in the haptic scene. If there has been a collision, then the intersected primitive suggests a surface contact point for the PHANTOM probe to move towards. In this case it is now necessary to perform an "intersection" test to determine if the line segment from the last position of the PHANTOM probe to the suggested surface contact point intersects any of the geometry in the scene (including the primitive with which there was a "collision").

The timings (in milliseconds) shown in Tables 1–5 were obtained by replaying the test data set on a four-processor 400 MHz PC, with 1 GB of physical memory. Each timing was obtained using only one processor. For comparison, we ran the same suite of tests on a single processor 300 MHz Pentium Pro with 128 MB memory. The hybrid approach appeared to be the most favorable on this system as well.

Table 1
Timings in milliseconds for Man Symbol, 5K triangles

Method	Adapt grid	Hybrid	OBBTree	GHOST
Ave Col. Hit	0.0122	0.00883	0.0120	0.0917
Worst Col. Hit	0.157	0.171	0.0800	0.711
Ave Col. Miss	0.00964	0.00789	0.00856	0.0217
Worst Col. Miss	0.0753	0.0583	0.0683	0.663
Ave Int. Hit	0.0434	0.0467	0.0459	0.0668
Worst Int. Hit	0.108	0.102	0.0793	0.100
Ave Int. Miss	0.0330	0.0226	0.0261	0.0245
Worst Int. Miss	0.105	0.141	0.0890	0.364
Ave. Query	0.019	0.014	0.017	0.048

Table 2
Timings in milliseconds for Man with Hat, 7K triangles

Method	Adapt grid	Hybrid	OBBTree	GHOST
Ave Col. Hit	0.0115	0.0185	0.0109	0.131
Worst Col. Hit	0.142	0.213	0.138	0.622
Ave Col. Miss	0.0104	0.00846	0.0101	0.0176
Worst Col. Miss	0.0800	0.0603	0.0813	0.396
Ave Int. Hit	0.0583	0.0568	0.0652	0.0653
Worst Int. Hit	0.278	0.200	0.125	0.233
Ave Int. Miss	0.0446	0.0237	0.0349	0.0322
Worst Int. Miss	0.152	0.173	0.111	0.287
Ave. Query	0.030	0.025	0.028	0.070

6.2. Comparison between algorithms

Since the algorithms run on a real-time system, we are not only interested in the average performance, but also in the worst case performance. Tables 1–5 show the timings in milliseconds obtained for both cases on each model and each contact configuration.

First of all, we noticed that our analysis of expected constant time performance for our hybrid collision detection algorithm is validated empirically with the timings given in Tables 1–5. With an exception of average query time for Table 2 where Assumptions 1 and 2 do not hold, the average query time was consistently in the range of 0.014 to 0.016 milliseconds, independent of the complexity of the model. In comparison, the average query time for native *GHOST* collision detection ranged from 0.048 to

Table 3
Timing in milliseconds for NanoSurface, 12K triangles

Method	Adapt grid	Hybrid	OBBTree	GHOST
Ave Col. Hit	0.0138	0.0101	0.0134	0.332
Worst Col. Hit	0.125	0.168	0.0663	0.724
Ave Col. Miss	0.00739	0.00508	0.00422	0.0109
Worst Col. Miss	0.0347	0.0377	0.0613	0.210
Ave Int. Hit	0.0428	0.0386	0.0447	0.0851
Worst Int. Hit	0.0877	0.102	0.0690	0.175
Ave Int. Miss	0.0268	0.0197	0.0213	0.0545
Worst Int. Miss	0.0757	0.0697	0.0587	0.284
Ave. Query	0.022	0.016	0.039	0.18

Table 4
Timings in milliseconds for Bronco, 18K triangles

Method	Adapt grid	Hybrid	OBBTree	GHOST
Ave Col. Hit	0.0113	0.00995	0.0125	0.104
Worst Col. Hit	0.136	0.132	0.177	0.495
Ave Col. Miss	0.0133	0.00731	0.0189	0.0280
Worst Col. Miss	0.128	0.0730	0.137	0.641
Ave Int. Hit	0.0566	0.0374	0.609	0.0671
Worst Int. Hit	0.145	0.105	0.170	0.293
Ave Int. Miss	0.0523	0.0225	0.0452	0.0423
Worst Int. Miss	0.132	0.133	0.167	0.556
Ave. Query	0.027	0.014	0.028	0.048

0.32 milliseconds, as the model size increased from 5K polygons to 79K polygons. Note that even with incoherent motions and varied triangle distribution, the average query time in Table 2 (0.025 milliseconds) is still well within the operational range of 1 millisecond desired by haptic display.

All of our algorithms are able to perform collision queries at rates faster than the required 1000 Hz force update rate for *all* the models we tested in the worst case. Although the hybrid approach often outperforms the algorithm based on OBBTrees, it is sometimes slightly slower than the algorithm based on OBBTrees. We conjecture that this behavior is due to the cache size of the CPU (independent of the memory size) and memory paging algorithm of the operating system. Among techniques that use hierarchical representations, cache access patterns can often have a dramatic impact on run time performance.

Table 5
Timings in milliseconds for Butterfly, 79K triangles

Method	Adapt grid	Hybrid	OBBTree	GHOST
Ave Col. Hit	0.0232	0.0204	0.0163	1.33
Worst Col. Hit	0.545	0.198	0.100	5.37
Ave Col. Miss	0.00896	0.00405	0.00683	0.160
Worst Col. Miss	0.237	0.139	0.121	3.15
Ave Int. Hit	0.228	0.0659	0.0704	0.509
Worst Int. Hit	0.104	0.138	0.103	1.952
Ave Int. Miss	0.258	0.0279	0.0256	0.229
Worst Int. Miss	0.0544	0.131	0.0977	3.28
Ave. Query	0.030	0.016	0.016	0.320

The hybrid approach requires more memory and is likely to have a less cachefriendly memory access pattern than the algorithm purely based on OBBTrees, despite the fact that both were well within the realm of physical memory available to the machine. Furthermore, by partitioning polygons into groups using grids, the hybrid technique can enable real-time local surface modification.

The adaptive grids-hashing scheme, a commonly used technique in ray-tracing, did not perform equally well in all cases. Once again, our hypothesis is that its inferior worst case behavior is due to its cache access patterns, in addition to its storage requirements. While the native *GHOST* method is competitive for the smaller model sizes, its performance does not scale as well for larger models. Our hybrid approach and our algorithm purely based on OBBTrees and the specialized overlap test appear to be relatively unaffected by the model complexity.

6.3. Dealing with deformations

The algorithmic framework of our collision detection system can also handle local deformation. Based on our prototype system implementation of *H-Collide* [18], we developed an interactive multiresolution modeling and 3D painting system using a haptic interface, called *inTouch* [17]. The contact information output by H-Collide is used for both model editing and painting. The data structure of H-Collide allows local modification of data structures easily and enables fast model deformation at interactive rates. Although the environment (in this case the model being deformed and reshaped) of *inTouch* is no longer static and Assumption 4 does not hold, H-Collide is still able to keep up with the KHz haptic update rates.

7. Conclusion

We have presented a fast and accurate collision detection algorithm for haptic interaction with polygonal models and a resulting system, H-COLLIDE, consisting of a suite of algorithms and a software

implementation. The algorithmic design may be extended to support a future 6-DoF haptic devices to perform collision tests between a pair of 3D objects. In addition, it can be combined with the tracing algorithm [41] to handle complex sculptured models more efficiently, using their control points.

Our ultimate goal is to support haptic interaction with complex CAD models for virtual prototyping applications. We plan to continue studying the cache behavior of the hierarchical representations, to design new hierarchy construction methods to allow even faster local modification of surfaces, and to work on seamless integration of algorithmic techniques and data structures to support a wide range of haptic devices.

Acknowledgements

We are grateful to Army Research Office, National Science Foundation, National Institute of Health National Center for Research Resources and Intel Corporation for their support and the reviewers for their comments.

References

- [1] J. Arvo, D. Kirk, A survey of ray tracing acceleration techniques, in: *An Introduction to Ray Tracing*, 1989, pp. 201–262.
- [2] R.S. Avila, L.M. Sobierajski, A haptic interaction method for volume visualization, in: *Proceedings of Visualization'96*, 1996, pp. 197–204.
- [3] D. Baraff, Curved surfaces and coherence for non-penetrating rigid body simulation, *ACM Comput. Graphics* 24 (4) (1990) 19–28.
- [4] G. Barequet, B. Chazelle, L. Guibas, J. Mitchell, A. Tal, Bboxtree: A hierarchical representation of surfaces in 3D, in: *Proc. of Eurographics'96*, 1996.
- [5] N. Beckmann, H. Kriegel, R. Schneider, B. Seeger, The r^* -tree: An efficient and robust access method for points and rectangles, in: *Proc. SIGMOD Conf. on Management of Data*, 1990, pp. 322–331.
- [6] F.P. Brooks, Jr., M. Ouh-Young, J.J. Batter, P.J. Kilpatrick, Project GROPE – Haptic displays for scientific visualization, *ACM Comput. Graphics* 24 (1990) 177–185.
- [7] S. Cameron, Approximation hierarchies and s -bounds, in: *Symposium on Solid Modeling Foundations and CAD/CAM Applications*, Proceedings, Austin, TX, 1991, pp. 129–137.
- [8] S. Cameron, A comparison of two fast algorithms for computing the distance between convex polyhedra, *IEEE Trans. Robotics Autom.* 13 (6) (1996) 915–920.
- [9] J. Cohen, M. Lin, D. Manocha, M. Ponamgi, I-Collide: An interactive and exact collision detection system for large-scale environments, in: *Proc. of ACM Interactive 3D Graphics Conference*, 1995, pp. 189–196.
- [10] J.E. Colgate, J.M. Brown, Factors affecting the Z-width of a haptic display, in: *IEEE Conference on Robotics and Automation*, 1994, pp. 3205–3210.
- [11] J.E. Colgate et al., Issues in the haptic display of tool use, in: *Proceedings of the ASME Haptic Interfaces for Virtual Environment and Teleoperator Systems*, 1994, pp. 140–144.
- [12] T.H. Corman, C.E. Leiserson, R.L. Rivest, *Introduction to Algorithms*, McGraw-Hill, 1992.
- [13] M. Finch, M. Falvo, V.L. Chi, S. Washburn, R.M. Taylor, R. Superfine, Surface modification tools in a virtual environment interface to a scanning probe microscope, in: P. Hanrahan and J. Winget (Eds.), *1995 Symposium on Interactive 3D Graphics*, ACM SIGGRAPH, April 1995, pp. 13–18.
- [14] S. Gibson, Beyond volume rendering: Visualization, haptic exploration, and physical modeling of element-based objects, in: *Proc. Eurographics Workshop on Visualization in Scientific Computing*, 1995, pp. 10–24.

- [15] E.G. Gilbert, D.W. Johnson, S.S. Keerthi, A fast procedure for computing the distance between objects in three-dimensional space, *IEEE J. Robotics Autom.* 4 (1988) 193–203.
- [16] S. Gottschalk, M. Lin, D. Manocha, ObbTree: A hierarchical structure for rapid interference detection, in: *Proc. of ACM SIGGRAPH'96*, 1996, pp. 171–180.
- [17] A. Gregory, S. Ehmann, M.C. Lin, inTouch: Interactive multiresolution modeling and 3D painting with a haptic interface, Technical Report, Department of Computer Science, University of North Carolina, 1999.
- [18] A. Gregory, M. Lin, S. Gottschalk, R. Taylor, H-Collide: A framework for fast and accurate collision detection for haptic interaction, in: *Proceedings of Virtual Reality Conference 1999*, 1999.
- [19] M. Held, J.T. Klosowski, J.S.B. Mitchell, Evaluation of collision detection methods for virtual reality fly-throughs, in: *Canadian Conference on Computational Geometry*, 1995.
- [20] P.M. Hubbard, Interactive collision detection, in: *Proceedings of IEEE Symposium on Research Frontiers in Virtual Reality*, October 1993.
- [21] D. Johnson, E. Cohen, A framework for efficient minimum distance computation, in: *IEEE Conference on Robotics and Automation*, 1998, pp. 3678–3683.
- [22] J. Klosowski, M. Held, J.S.B. Mitchell, K. Zikan, H. Sowizral, Efficient collision detection using bounding volume hierarchies of k -DOPs, *IEEE Trans. Visual. Comput. Graph.* 4 (1) (1998) 21–36.
- [23] J. Klosowski, M. Held, J.S.B. Mitchell, H. Sowizral, K. Zikan, Efficient collision detection using bounding volume hierarchies of k -dops, in: *SIGGRAPH'96 Visual Proceedings*, 1996, p. 151.
- [24] S. Krishnan, A. Pattekar, M. Lin, D. Manocha, Spherical shell: A higher order bounding volume for fast proximity queries, in: *Proc. of Third International Workshop on Algorithmic Foundations of Robotics*, 1998, pp. 122–136.
- [25] E. Larsen, S. Gottschalk, M. Lin, D. Manocha, Fast proximity queries with swept sphere volumes, Technical Report TR99-018, Department of Computer Science, University of North Carolina, 1999.
- [26] M. Lin, S. Gottschalk, Collision detection between geometric models: A survey, in: *Proc. of IMA Conference on Mathematics of Surfaces*, 1998.
- [27] M.C. Lin, J.F. Canny, Efficient algorithms for incremental distance computation, in: *IEEE Conference on Robotics and Automation*, 1991, pp. 1008–1014.
- [28] W. Mark, S. Randolph, M. Finch, J. Van Verth, R.M. Taylor II, Adding force feedback to graphics systems: Issues and solutions, in: H. Rushmeier (Ed.), *SIGGRAPH'96 Conference Proceedings*, Annual Conference Series, 1996, pp. 447–452.
- [29] T.M. Massie, J.K. Salisbury, The phantom haptic interface: A device for probing virtual objects, in: *Proc. of ASME Haptic Interfaces for Virtual Environment and Teleoperator Systems 1* (1994) 295–301.
- [30] W. McNeely, K. Puterbaugh, J. Troy, Six-degree-of-freedom haptic rendering using voxel sampling, in: *Proceedings of ACM SIGGRAPH'99*, 1999, pp. 401–408.
- [31] A. Nahvi, D. Nelson, J. Hollerbach, D. Johnson, Haptic manipulation of virtual mechanisms from mechanical CAD designs, in: *Proc. of 1998 Conference on Robotics and Automation*, 1998, pp. 375–380.
- [32] B. Naylor, J. Amanatides, W. Thibault, Merging BSP trees yield polyhedral modeling results, in: *Proc. of ACM SIGGRAPH*, 1990, pp. 115–124.
- [33] M. Ouh-Young, Force display in molecular docking, Ph.D. Thesis, University of North Carolina, Computer Science Department, 1990.
- [34] M.H. Overmars, Point location in fat subdivisions, *Inform. Proc. Lett.* 44 (1992) 261–265.
- [35] S. Quinlan, Efficient distance computation between non-convex objects, in: *Proceedings of International Conference on Robotics and Automation*, 1994, pp. 3324–3329.
- [36] D.C. Ruspini, K. Kolarov, O. Khatib, The haptic display of complex graphical environments, in: *Proc. of ACM SIGGRAPH*, 1997, pp. 345–352.
- [37] K. Salisbury, D. Brock, T. Massie, N. Swarup, C. Zilles, Haptic rendering: Programming touch interaction with virtual objects, in: *Proc. of 1995 ACM Symposium on Interactive 3D Graphics*, 1995, pp. 123–130.

- [38] H. Samet, Spatial Data Structures: Quadtree, Octrees and Other Hierarchical Methods, Addison-Wesley, Reading, MA, 1989.
- [39] SensAble Technologies, Inc., *GHOSTTM*: Software developer's toolkit, Programmer's Guide, 1997.
- [40] R.M. Taylor, W. Robinett, V.L. Chii, F. Brooks, W. Wright, The nanoManipulator: A virtual-reality interface for a scanning tunneling microscope, in: Proc. of ACM SIGGRAPH, 1993, pp. 127–134.
- [41] T.V. Thompson, D. Johnson, E. Cohen, Direct haptic rendering of sculptured models, in: Proc. of ACM Interactive 3D Graphics, 1997, pp. 167–176.